arXiv:2110.03655v3 [cs.LG] 30 Jun 2022

Augmenting Reinforcement Learning with Behavior Primitives for Diverse Manipulation Tasks

Soroush Nasiriany[1], Huihan Liu[1], Yuke Zhu[1]

***Abstract*— Realistic manipulation tasks require a robot to interact with an environment with a prolonged sequence of motor actions. While deep reinforcement learning methods have recently emerged as a promising paradigm for automating manipulation behaviors, they usually fall short in long-horizon tasks due to the exploration burden. This work introduces Manipulation Primitive-augmented reinforcement Learning (MAPLE), a learning framework that augments standard reinforcement learning algorithms with a pre-defined library of behavior primitives. These behavior primitives are robust functional modules specialized in achieving manipulation goals, such as grasping and pushing. To use these heterogeneous primitives, we develop a hierarchical policy that involves the primitives and instantiates their executions with input parameters. We demonstrate that MAPLE outperforms baseline approaches by a significant margin on a suite of simulated manipulation tasks. We also quantify the compositional structure of the learned behaviors and highlight our method's ability to transfer policies to new task variants and to physical hardware. Videos and code are available at `https://ut-austin-rpl.github.io/maple`**

I. Introduction

Enabling autonomous robots to solve diverse and complex manipulation tasks has been a grand challenge for decades. In recent years, deep reinforcement learning (DRL) approaches have made great strides towards designing robot manipulation behaviors that are difficult to engineer manually [27, 28, 50, 51]. Nonetheless, state-of-the-art DRL models fall short in long-horizon tasks due to the exploration challenge — the robot has to explore a prohibitively large space of possible behaviors for accomplishing a task. To remedy the exploration burden, prior DRL work has developed various temporal abstraction frameworks to exploit the hierarchical structure of manipulation tasks [4, 12, 43, 57]. These methods learn low-level controllers, often modeled as *skills* or *options*, together with high-level controllers from trial-and-error. While they have demonstrated greater scalability than vanilla DRL methods, they often suffer from high sample complexity, lack of interpretability, and brittle generalization.

In the meantime, decades-long research in robotics has developed a rich repertoire of functional modules specialized at particular robot behaviors, such as grasping [7] and motion planning [23, 29]. These pre-built functional modules, which we refer to as *behavior primitives*, exhibit a high degree of robustness and reusability for achieving certain manipulation goals, such as picking up objects with the end-effector and moving the robot to a target configuration in a collision-free path. In spite of their specialties, it remains a challenge for DRL algorithms to use them as the building blocks to scaffold complex tasks. The challenge is primarily due to the fact that these behavior primitives are *heterogeneous* by design. They take non-uniform parameters as input, operate at varying temporal resolutions, and exhibit distinct behaviors. This thus requires an algorithm to reason about the temporal decomposition of a complex task and adaptively compose these behavior primitives accordingly.

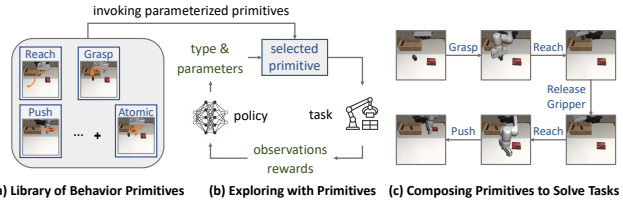


Fig. 1: **Overview of MAPLE.** (a) We present a learning framework that augments the robot's atomic motor actions with a library of versatile behavior primitives. (b) Our method learns to compose these primitives via reinforcement learning. (c) This enables the agent to solve complex long-horizon manipulation tasks.

A variety of hierarchical modeling approaches in robotics have used behavior modules as low-level building blocks. Notably, task-and-motion planning [17, 26, 66] and neural programming [22, 71] methods have used primitives such as motion planners and pick-and-place controllers to model manipulation tasks in a compositional fashion. They require well-specified domain knowledge to perform task planning or strong human supervision to train a high-level controller with ground-truth task decomposition. These assumptions limit the scalability of these methods in realistic tasks.

In this work, we introduce MAPLE (**Ma**nipulation **P**rimitive-augmented reinforcement **Le**arning), a general DRL algorithm that harnesses a set of pre-built behavior primitives for solving long-horizon manipulation tasks. To address the exploration challenge of DRL algorithms, our method uses a library of high-level behavior primitives (such as grasping or pushing objects) in conjunction with low-level motor actions to autonomously learn a hierarchical policy (see Fig. 1). Our algorithm models each behavior primitive as an implementation-agnostic controller that produces a temporally extended behavior. At a given state, our DRL policy invokes a behavior primitive (or an atomic motor action) and instantiates it with input parameters. For example, the input parameters to a 6-DoF grasping module can be the pre-grasp end-effector pose. The selected primitive interprets the input parameters and executes one or a sequence of motor actions to realize its specialized behavior. By integrating behavior primitives into DRL algorithms, MAPLE shields away a substantial portion of complexity in manipulation planning,

[1]The University of Texas at Austin. Correspondence: `soroush@cs.utexas.edu`

while leaving the flexibility to a generic reinforcement learning algorithm to discover the compositional structure of tasks without strong domain knowledge. Furthermore, by retaining low-level motor actions MAPLE can rely on these actions for the stages of tasks where the finite library of behavior primitives is insufficient to express a desired behavior.

We conduct an extensive set of experiments on a suite of eight manipulation tasks of varying complexities in the robosuite simulation framework [74]. We compare our method to standard DRL approaches [19] that only use low-level motor actions, hierarchical DRL methods that learn options [8, 43, 73] or open-loop task schemas [8]. MAPLE achieves a 70% increase in task success rate compared to using only atomic actions, becoming the only method that consistently solved all single-arm tasks in the standard robosuite benchmark. We also devise a data-driven metric to quantitatively examine the compositionality of manipulation tasks contingent on the available primitives, offering new insight on the challenges and opportunities of compositional modeling for realistic manipulation tasks.

We highlight three contributions of this work: 1) We develop a novel method that augments standard DRL algorithms with pre-defined behavior primitives to reduce the exploration burden; 2) We validate the effectiveness of our method in solving diverse manipulation tasks and quantitatively analyze the compositional structure of these tasks; and 3) We show that the modularity and abstraction offered by the behavior primitives facilitate knowledge transfer of the learned policies to new task variants and physical hardware.

II. Related Work

Deep Reinforcement Learning. Prior work on DRL has investigated a number of approaches to solve long-horizon tasks, through improved exploration strategies [6, 21, 53, 54], learning options [4, 5, 43, 63, 73], unsupervised skill discovery [11, 59], and integrating planning [12, 47]. Despite these efforts, today's DRL methods still struggle in long-horizon robotic tasks due to the exploration burden of learning from scratch. Recent work has examined the use of offline data to alleviate the exploration burden in DRL, through demonstration-guided RL [18, 46, 56], learned behavioral priors [55, 62] and action spaces [1, 2] from demonstrations, and offline RL [15, 16, 33, 40]. While promising, these methods can be difficult to scale up due to the costs of acquiring offline data.

Hierarchical Modeling in Robotics. Outside of DRL, there has been a plethora of work in robotics dedicated to building customized functional modules that emit specific robot behaviors, such as grasping [7, 38] and motion planning [3, 29]. Prior works on task-and-motion planning [17, 26, 66] and neural programming [22, 71] have developed hierarchical models that leverage these modules as building blocks to scaffold manipulation tasks. While these methods have demonstrated impressive capabilities in restrictive domains, their applicability has been limited by their reliance on domain knowledge or human supervision.

To bridge the gap between these models and DRL algorithms that learn from scratch, recent work has harnessed prebuilt primitives, such as model-based planners [34], motion planners [68, 72], movement primitives [23, 48], and prebuilt skills [8, 35, 60, 61, 64], to expedite DRL algorithms. These approaches aim at retaining the flexibilities of RL algorithms while benefiting from the temporal abstraction provided by the primitives. However, these works are limited as they are confined to using only one or two specific primitives [34, 68, 72], employ rigid primitives that are not reconfigurable [60, 64], or hard-code how the primitives are composed [61]. In contrast, our method adopts a set of versatile primitives to solve diverse manipulation tasks.

Reinforcement Learning with PAMDPs. Our formalism specifically falls under the established reinforcement learning framework of Parameterized Action MDPs (PAMDPs) [41], in which the agent executes a parameterized primitive at each decision-making step. We note that several prior works [13, 20, 24, 67, 69] have adapted off-the-shelf deep RL algorithms to the PAMDP setting. Nonetheless, they have focused on relatively simple game domains, shielding away practical challenges in robot manipulation, such as high-dimensional continuous state/action spaces and heterogeneous primitives. We note that Chitnis et al. [8] and Lee et al. [35] have modeled robot manipulation with PAMDPs. We provide empirical comparisons to demonstrate the limitations of their modeling choices, yielding less competitive performance in challenging manipulation tasks than ours. Concurrent work by Dalal et al. [9] also studies the application of robotic primitives for manipulation tasks. Our work complements theirs with additional analysis on the compositional structure of the learned behavior and experiments demonstrating the ability to transfer learned policies to novel task variants and to physical hardware.

III. Method

Our goal is to enable robots to leverage behavior primitives to solve manipulation tasks effectively and efficiently. To that end, we seek a library of behavior primitives that serve as the building blocks to scaffold manipulation tasks and a reinforcement learning algorithm that composes these primitives to solve tasks. To evaluate whether our algorithm facilitates compositional behaviors, we also propose a metric to quantify the degree to which the resulting learned behavior is compositional. See Fig. 1 for an overview of our method.

A. Decision-Making with Parameterized Behavior Primitives

We adopt reinforcement learning (RL) as the underlying decision-making framework. The objective of RL is to maximize the expected infinite sum of discounted rewards in a Markov Decision Process (MDP), defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, p, p_0, \gamma)$. The entities in the tuple represent the state space, the action space, the reward function, the transition function, the initial state distribution, and the discount factor. In most robotic RL problems, the action space $\mathcal{A}$ consists of all atomic actions $u \in \mathbb{R}^{d_{\text{control}}}$ provided by the robot, such as end-effector displacements. We augment this action

space with a heterogeneous library of behavior primitives $\mathcal{L} = \{a^1, a^2, \cdots, a^k\}$ that perform semantically meaningful behaviors. Formally, each behavior primitive $a \in \mathcal{L}$ is represented by a control module $M_a(x)$ that executes a finite, variable sequence of atomic actions $(u_1, u_2, \cdots, u_t), u_i \in \mathbb{R}^{d_{\text{control}}}$, where the exact action sequences are specified by input parameters $x \in \mathbb{R}^{d_a}$. Here d_a is the dimension of the input parameters to the primitive a that varies across different primitives. To incorporate these behavior primitives, we recast our decision-making problem as a Parameterized Action MDP (PAMDP) [41], where at each decision-making step the agent executes a parameterized action $(a, x) \in \mathcal{A}$ consisting of the type of primitive a and its parameters x.

B. Behavior Primitives: Building Blocks for Manipulation

We are interested in equipping agents with a library of versatile primitives that serve as the core building blocks for diverse manipulation tasks. Our decision-making algorithm assumes no knowledge of the implementations of these primitives; these primitives can comprise closed-loop skills learned via reinforcement [19, 58] or imitation learning [52], analytical motion planners [29], and even full-fledged grasping systems [7, 38]. Regardless of their inner workings, we must ensure that our primitives are versatile and adaptive to behavioral variations. In our learning framework, we consider these primitives as *functional APIs* that take input parameters x that instantiate action execution. The input parameters usually have clear semantics, such as the 6-DoF end-effector pose for a grasping primitive or a target robot configuration for a motion planning primitive. We recognize that our library of primitives may still not be universally applicable in every setting, and equipping the agent solely with these primitives may limit the set of possible behaviors that the agent can achieve. We address this limitation by introducing an additional *atomic primitive* a^{atom} dedicated to performing atomic robot actions to fill in any missing gaps that cannot be fulfilled by the other primitives.

In this work, we design a library of five primitives, including prehensile and non-prehensile motions, that forms the basis for many manipulation tasks:

- **Reaching:** The robot moves its end-effector to a target location (x, y, z), specified by the input parameters. Execution takes at most 15 atomic actions.
- **Grasping:** The robot moves its end-effector to a pre-grasp location (x, y, z) at a yaw angle ψ, specified by the input parameters, and closes its gripper. Execution takes at most 20 atomic actions.
- **Pushing:** The robot reaches a starting location (x, y, z) at a yaw angle ψ and then moves its end-effector by a displacement $(\delta_x, \delta_y, \delta_z)$. The input parameters are 7D. Execution takes at most 20 atomic actions.
- **Gripper Release:** The robot repeatedly applies atomic actions to open its gripper. This primitive has no input parameters. Execution takes 4 atomic actions.
- **Atomic:** The robot applies an atomic action $\in \mathbb{R}^{d_{\text{control}}}$.

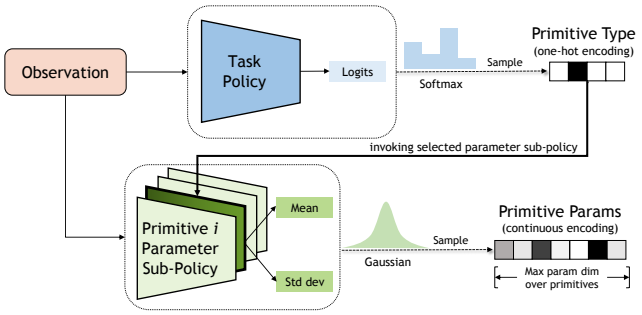


Fig. 2: **Policy Architecture.** We adopt a hierarchical policy, with a high-level task policy that determines *which* primitive to apply and a low-level parameter policy that determines *how* to instantiate that primitive.

We implemented these primitives as hard-coded closed-loop controllers, each requiring only a handful of lines of code. We highlight that these primitives take input parameters of different dimensions, operate at variable temporal lengths, and produce distinct behaviors. These properties make them challenging to utilize in a learning framework. In the following, we will introduce our algorithm for composing these primitives to solve diverse manipulation tasks.

C. Composing Primitives via Reinforcement Learning

We follow the PAMDP framework outlined in Section III-A. Previous work has explored various policy structures that reason over parameterized primitives. The simplest approach is a flat policy [20, 68] that outputs a distribution over the primitive type a and all primitive parameters $\{x^1, x^2, \cdots, x^k\}$. A major drawback of this approach is that the total number of policy outputs can quickly become intractable as additional primitives are introduced. We address this limitation with a hierarchical policy where at the high level a *task policy* π_{tsk} determines the primitive type a and at the low level a *parameter policy* π_p determines the corresponding primitive parameters x. For implementation, we represent the task policy as a single neural network and the parameter policy as a collection of sub-networks, with one sub-network dedicated for each primitive. This enables us to accommodate primitives with heterogeneous parameterizations. To allow batch tensor computations across primitives with different parameter dimensions, these parameter policy sub-networks all output a "one size fits all" distribution over parameters $x \in \mathbb{R}^{d_A}$, where $d_A = \max_a d_a$ is the maximum parameter dimension over all primitives. At primitive execution we simply truncate the parameters x to the length d_a of the chosen primitive a. See Fig. 2 for an illustration of our policy architecture. In addition to reducing the overall number of output parameters our hierarchical design facilitates modular reasoning, delegating the high-level to focus on *which* primitive to execute and the low-level to focus on *how* to instantiate that primitive. We note that a few prior works have previously explored this hierarchical design [13, 67] but to our knowledge we are the first to demonstrate its utility on complex manipulation domains with a set of heterogeneous primitives.

In principle, we can integrate our policy architecture with any DRL algorithm designed for continuous control;

we choose Soft Actor-Critic (SAC) [19]. We modify the standard critic neural network $Q_\theta(s,a)$ and actor neural network $\pi_\phi(a|s)$ with our critic network $Q_\theta(s,a,x)$ and our hierarchical policy networks $\pi_{tsk_\phi}(a|s)$, $\pi_{p_\psi}(x|s,a)$. The losses for the critic, task policy, and parameter policy are defined respectively (with components pertaining to the task policy in red and the parameter policy in blue):

$$J_Q(\theta) = \Big(Q_\theta(s,a,x) - \Big(r(s,a,x) + \gamma\Big(Q_{\bar{\theta}}(s',a',x') - \alpha_{tsk}\log(\pi_{tsk_\phi}(a'|s')) - \alpha_p \log(\pi_{p_\psi}(x'|s',a'))\Big)\Big)\Big)^2$$

$$J_{\pi_{tsk}}(\phi) = \mathop{\mathbb{E}}_{a\sim\pi_{tsk_\phi}}\Big[\alpha_{tsk}\log(\pi_{tsk_\phi}(a|s)) - \mathop{\mathbb{E}}_{x\sim\pi_{p_\psi}} Q_\theta(s,a,x)\Big]$$

$$J_{\pi_p}(\psi) = \mathop{\mathbb{E}}_{a\sim\pi_{tsk_\phi}}\mathop{\mathbb{E}}_{x\sim\pi_{p_\psi}}\Big[\alpha_p\log(\pi_{p_\psi}(x|s,a)) - Q_\theta(s,a,x)\Big]$$

Here α_{tsk} and α_p control the maximum entropy objective for the task policy and parameter policy, respectively.

D. Facilitating Exploration with Affordances

Compared with existing methods that reason purely over atomic actions, our algorithm benefits from accelerated exploration due to the temporal abstraction provided by our behavior primitives. However, even reasoning with temporally extended actions can present an exploration challenge [55]. One way to address this issue is to equip the agent with *affordances* that help to discern the utility of actions in different settings. For example, a grasping skill is only appropriate when applied in the vicinity of graspable objects, and a pushing skill is only appropriate in the vicinity of pushable objects. In our framework, these affordances can be expressed by adding to the reward function an auxiliary affordance score $s_{\text{aff}}(s,x;a) \in [0,1]$ that measures the affinity for parameters x at a particular state s for a given primitive a. These affordances scores can in principle come from learned models trained on robot interaction data [39, 42, 45, 61, 70] or human data [10, 14, 32, 44]. Nonetheless, as our primitive parameters carry clear semantic meanings, we can analytically define these affordance scores based on the objects' physical states. Concretely, for the *atomic* and *gripper release* primitives, we always give an affordance score of 1 to enable the universal applicability of these primitives. For the remaining *reach*, *grasp*, and *push* primitives we implement general, easy-to-define affordances encouraging the agent to reach relevant areas of interest in the workspace. More specifically, these primitives all involve reaching a location $x_{reach} = \texttt{[x[0], x[1], x[2]]}$ and we encourage the agent to specify the reaching parameters x_{reach} to be within a threshold of a set of keypoints P:

$$s_{\text{aff}}(s,x;a) = \max_{p\in P} \; 1 - \tanh\Big(\max(\|x_{reach} - p\| - \tau, 0)\Big)$$

The keypoints P are the locations of objects to push for the push primitive, the locations of objects to grasp for the grasp primitive, and the locations of reaching targets for the reach primitive.

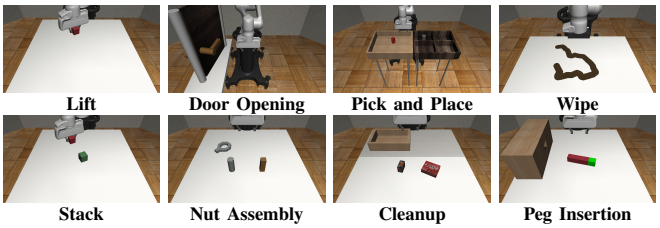


Fig. 3: **Simulated Environments.** We perform evaluations on eight manipulation tasks. The first six come from the robosuite benchmark [74]. We designed the last two to test our method in multi-stage, contact-rich tasks: Cleanup requires storing a spam can into a storage bin and a jello box at a corner; Peg Insertion requires inserting a peg into a block.

E. Quantifying Compositionality

Our framework is based upon the hypothesis that most manipulation tasks have an intrinsic compositional structure and that our algorithm can discover this structure. While there is no explicit mechanism to discover recurring patterns of primitives, our algorithm exhibits compositional reasoning by preferring the use of high-level primitives over low-level ones. Due to the temporal abstraction encapsulated by the high-level primitives, the agent can make far greater progress toward solving the task by using high-level primitives and thus receives higher average reward per timestep. This incentivizes the agent to choose higher-level primitives over lower-level actions whenever appropriate. We additionally examine the degree to which our learned agent exhibits compositional behaviors with a quantifiable metric. Assuming a set of trajectories in which the agent solved a task $\mathcal{T}$: $\{\tau^i\}_{i=1}^n = \{(s_1^i, (a_1^i, x_1^i), \cdots, s_{T_i}^i, (a_{T_i}^i, x_{T_i}^i), s_{T_i+1}^i)\}_{i=1}^n$, the *task sketches* $\{K^i\}_{i=1}^n = \{(a_1^i, a_2^i, \cdots a_{T_i}^i)\}_{i=1}^n$ capture high-level task semantics and provide useful abstractions through which we can analyze the compositional structure of these trajectories. Intuitively, agents that demonstrate compositional reasoning will express recurring patterns of behaviors across their task sketches and prefer the use of high-level primitives over low-level ones. We quantify this intuition by computing the *Levenshtein distance* [36] among task sketches, which measures the minimum number of single-token edits needed to transform one task sketch to another. We represent each non-atomic primitive *type* as a unique token, and in order to explicitly discourage the use of low-level atomic actions, we represent each *individual occurrence* of an atomic primitive in our task sketches as a unique token. Given a task $\mathcal{T}$ and available primitives $\mathcal{L}$, we compute the compositionality of the agent's behavior as the average pairwise normalized score between the task sketches:

$$f_{comp}(\mathcal{T};\mathcal{L}) = \frac{1}{n(n-1)}\sum_{i\neq j} 1 - \frac{d_{\text{Lev}}(K_i,K_j)}{\max(|K_i|,|K_j|)} \quad (1)$$

IV. Experiments

In our experiments we study 1) whether our method can compose pre-built behavior primitives and atomic actions to solve complex tasks, 2) the degree to which the learned behavior is compositional, and 3) whether our approach is amenable to transfer to task variants and to real hardware.

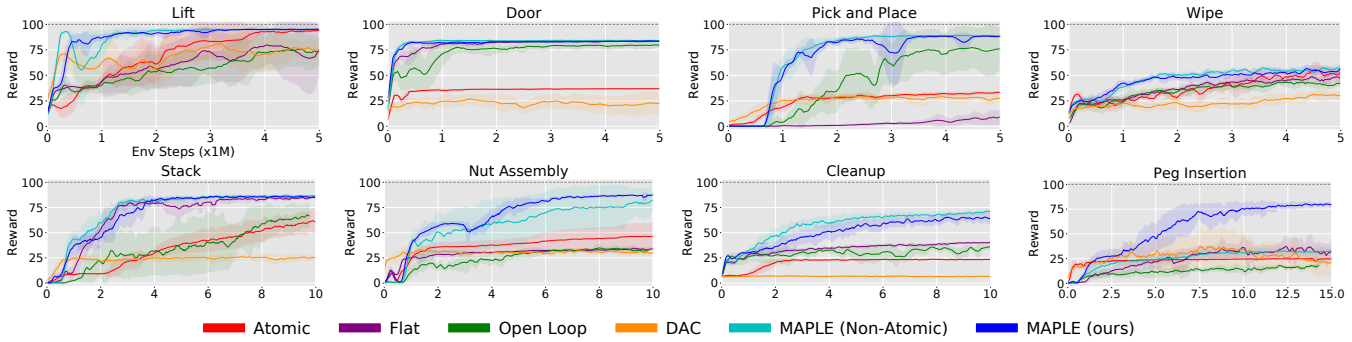


Fig. 4: **Main Results.** Learning curves showing average episodic task rewards throughout training, normalized between 0 and 100. All experiments are averaged over 5 seeds, with shaded regions depicting the standard deviation.

A. Experimental Setup

We examine these questions on robosuite [74], a framework for simulated robot manipulation tasks. We consider a comprehensive suite of eight manipulation tasks of varying complexities (see Fig. 3). We adopt a Franka Emika Panda robot arm controlled via operational space control (OSC) [30]. At each decision-making step the agent executes either an atomic OSC action or one of the non-atomic primitives outlined in Section III-B. In return the agent receives 1) a dense reward signal indicating task progress and 2) an observation comprising the robot's proprioceptive state and pose information of the objects in the environment.

B. Quantitative Evaluation

We compare our method (**MAPLE**) to five baselines. The first baseline uses exclusively atomic actions (**Atomic**), which corresponds to the standard Soft Actor-Critic model [19] trained on end-effector commands. To understand the effect of hierarchy on our policy design, we compare to a flat variant where the policy outputs the primitive type and parameters independently (**Flat**), following the design by Lee et al. [35] and Neunert et al. [49]. We also compare to a variant of our method using an open loop task policy (**Open Loop**), following Chitnis et al. [8] which suggests utilizing an open-loop task schema improves the sample efficiency of the algorithm. Next, we compare to Hierarchical Reinforcement learning with Off-policy correction (**HIRO**) [43] and Double Actor-Critic (**DAC**) [73], state-of-the-art hierarchical DRL methods which learn low-level policies (or options) along with high-level controllers. HIRO failed to make progress and we thus omit it from our results. Finally, we compare to a self baseline where we include all primitives *except* the atomic primitive (**MAPLE (Non-Atomic)**), to understand whether we need atomic actions to satisfy behaviors that cannot be fulfilled by the non-atomic primitives. All baselines using behavior primitives use the affordance score outlined in Section III-D.

Fig. 4 outlines environment rewards throughout training. We also evaluated the final task success rates at the end of training: MAPLE achieved the highest average success rate across all baselines (90%), compared to 19% for the Atomic baseline, 36% for Flat, 41% for Open Loop, 11% for DAC, and 79% for MAPLE (Non-Atomic). First, we see that the inclusion of non-atomic primitives allows MAPLE to significantly outperform the Atomic baseline, achieving on average 2-3$\times$ higher rewards and 71% higher success rate. Qualitatively we found that the Atomic baseline fails to advance past the first stage in most tasks while our method successfully solves all tasks. Next we find that the Flat baseline is unable to reliably solve all tasks, demonstrating that our hierarchical policy design is key to reasoning over a heterogeneous set of primitives. While the Open Loop baseline is able to solve basic tasks such as Door Opening and Pick and Place, it struggles with tasks that require the agent to adaptively reason about the current state of the task. DAC is only able to solve the Lift task, highlighting the difficulty of learning complex tasks *from scratch* even when employing temporal abstraction. Finally we find that the Non-Atomic self baseline is on par with our method in most tasks, yet it notably fails for Peg Insertion as the non-atomic primitives are not expressive enough to perform the contact-rich insertion phase. Together, these results highlight that given an appropriate primitive library and policy structure we can solve a wide range of manipulation tasks.

C. Model Analysis

Emergence of Compositional Structures. We present an analysis of the task sketches that our method learned for each task in Fig. 5. We see evidence that the agent unveils compositional task structures by applying temporally extended primitives whenever appropriate and relying on atomic actions otherwise. For example, for the peg insertion task the agent leverages the grasping primitive to pick up the peg and the reaching primitive to align the peg with the hole in the block, but then it uses atomic actions for the contact-rich insertion phase. In Fig. 5 we also quantify the compositionality of these task sketches via f_{comp} defined in Eq. (1). As expected, tasks involving contact interactions such as Peg Insertion and Wiping have lower scores than prehensile tasks such as Pick and Place and Stacking.

Transfer to Semantically Similar Task Variants. We have seen how task sketches enable interpretability by serving as blueprints of high-level semantic task structure. We can leverage these task sketches to accelerate learning on similar task instances. We propose to re-use the task sketch from a semantically similar task, and only learn the corresponding primitive parameters. We validate this idea on the Pick and

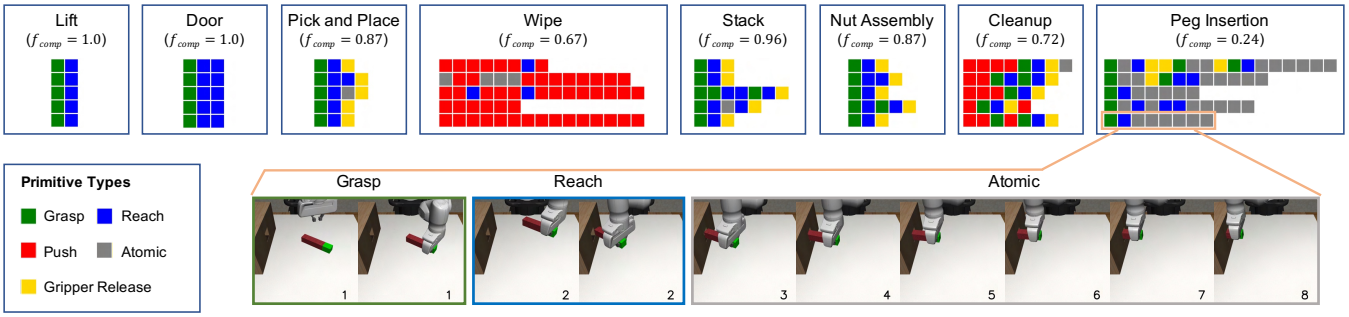
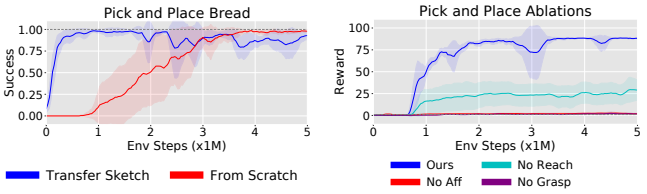


Fig. 5: **Analyzing Learned Behavior.** (Top) We visualize the task sketches that our agent has learned. Each row corresponds to a single sketch progressing temporally from left to right. For each task we also report the compositionality score f_{comp}. (Bottom) We visualize the behavior for a peg insertion sketch.



(a) Task Transfer (b) Ablations

Fig. 6: **(a) Task Transfer.** We transfer the learned task sketch from a source task (pick and place can) to a semantically similar task variant (pick and place bread), enabling us to learn the target task over $5\times$ faster. **(b) Ablations**. Without affordances, reaching, or grasping, the agent is unable to solve tasks due to the exploration burden.

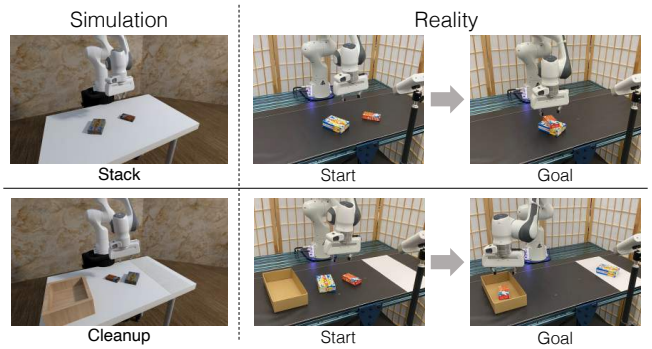


Fig. 7: **Transfer to Real-World Tasks.** We transfer our policy trained on simulated environments to the real-world Stack and Cleanup tasks.

Place domain, where we transfer the task sketch from a source task of placing a soda can into one bin, to a target task of placing a loaf of bread into a different bin. As shown in Fig. 6a, we solve the bread task significantly faster than learning the task from scratch with a sample efficiency of over $5\times$. This suggests that our task sketch serves as a high-level scaffold of a manipulation task, which can be re-used by learning algorithms for faster adaptation to related tasks.

Ablation Study. We perform an ablation study examining the role of affordances and individual manipulation primitives in facilitating exploration. We specifically perform experiments on the Pick and Place task, comparing our method (**Ours**) to ablations 1) without affordances in the reward function (**No Aff**), 2) without the reaching skill (**No Reach**), and 3) without the grasping skill (**No Grasp**). We see in Fig. 6b that without these components the agent fails to solve the task, underscoring that our method is reliant on the appropriate primitive skills and affordances to effectively overcome the exploration burden.

D. Real-World Evaluation

We conclude with an evaluation on real-world copies of the Stack and Cleanup tasks (see Fig. 7). As our behavior primitives offer high-level action abstractions and encapsulate low-level complexities of motor actuation, our policies can directly transfer to the real world. We trained MAPLE on simulated versions of these tasks and executed the resulting policies to the real world. We re-implemented our behavior primitives on the real robot and used an off-the-shelf pose estimation model [65] to estimate environment states. We successfully transferred the policy to the real world, with an average success rate of 93% on Stack and 83% on Cleanup.

V. Conclusion

We presented MAPLE, a reinforcement learning framework that incorporates behavior primitives in conjunction with low-level motor actions to solve complex manipulation tasks. Our experiments demonstrate that behavior primitives can significantly improve exploration while low-level motor actions allow us to retain flexibility to learn intricate behaviors. Our work opens the possibility for several avenues for future work. First, learning affordances using data-driven methods [32, 39, 42, 61] can expand the scalability of our method. Second, while atomic actions can help fill in gaps where the primitives are insufficient (such as peg insertion), we are unable to fill in large gaps that require a significant number of low-level action executions (as seen in the ablation experiments). Further research on exploration and credit assignment is needed to overcome these challenges. Finally, an exciting avenue for future work is to continually discover recurring compositions of primitives and add them to the library of primitives, which can ultimately enable curriculum learning of progressively more challenging tasks.

Acknowledgment

We would like to thank Yifeng Zhu for assisting with the real-world experiments and Abhishek Joshi for assisting with rendering content. We also thank Yifeng Zhu, Braham Snyder, and Josiah Wong for providing feedback on this manuscript. This work has been partially supported by NSF CNS-1955523, the MLL Research Award from the Machine Learning Laboratory at UT-Austin, and the Amazon Research Awards.

REFERENCES

[1] A. Ajay, A. Kumar, P. Agrawal, S. Levine, and O. Nachum, "Opal: Offline primitive discovery for accelerating offline reinforcement learning," in *ICLR*, 2021.

[2] A. Allshire, R. Martín-Martín, C. Lin, S. Manuel, S. Savarese, and A. Garg, "Laser: Learning a latent action space for efficient reinforcement learning," in *ICRA*, 2021.

[3] N. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *ICRA*, 1996.

[4] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *AAAI*, 2017.

[5] A. Bagaria and G. Konidaris, "Option discovery using deep skill chaining," in *ICLR*, 2020.

[6] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *NIPS*, 2016.

[7] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis—a survey," *IEEE Transactions on Robotics*, 2013.

[8] R. Chitnis, S. Tulsiani, S. Gupta, and A. Gupta, "Efficient bimanual manipulation using learned task schemas," in *ICRA*, 2020.

[9] M. Dalal, D. Pathak, and R. Salakhutdinov, "Accelerating robotic reinforcement learning via parameterized action primitives," in *NeurIPS*, 2021.

[10] T.-T. Do, A. Nguyen, and I. Reid, "Affordancenet: An end-to-end deep learning approach for object affordance detection," in *ICRA*, 2018.

[11] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," in *ICLR*, 2018.

[12] B. Eysenbach, R. Salakhutdinov, and S. Levine, "Search on the replay buffer: Bridging planning and reinforcement learning," in *NeurIPS*, 2019.

[13] Z. Fan, R. Su, W. Zhang, and Y. Yu, "Hybrid actor-critic reinforcement learning in parameterized action space," in *IJCAI*, 2019.

[14] K. Fang, T.-L. Wu, D. Yang, S. Savarese, and J. J. Lim, "Demo2vec: Reasoning object affordances from online videos," in *CVPR*, 2018.

[15] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, *D4rl: Datasets for deep data-driven reinforcement learning*, 2020.

[16] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *ICML*, 2019.

[17] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual Review of Control, Robotics, and Autonomous Systems*, 2021.

[18] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," in *CoRL*, 2019.

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.

[20] M. Hausknecht and P. Stone, "Deep reinforcement learning in parameterized action space," in *ICLR*, 2016.

[21] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," in *NIPS*, 2016.

[22] D.-A. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, and J. C. Niebles, "Neural task graphs: Generalizing to unseen tasks from a single video demonstration," in *CVPR*, 2019.

[23] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, 2013.

[24] A. Jain, A. Szot, and J. J. Lim, "Generalization to new actions in reinforcement learning," in *ICML*, 2020.

[25] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *ICLR*, 2017.

[26] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, 2011.

[27] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," in *CoRL*, 2018.

[28] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman, "Mt-opt: Continuous multi-task robotic reinforcement learning at scale," *arXiv*, 2021.

[29] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, 2011.

[30] O. Khatib, "Inertial properties in robotic manipulation: An object-level framework," *IJRR*, 1995.

[31] M. Klissarov, P.-L. Bacon, J. Harb, and D. Precup, *Learning options end-to-end for continuous action tasks*, 2017. arXiv: 1712.00004 [cs.LG].

[32] M. Kokic, D. Kragic, and J. Bohg, "Learning task-oriented grasping from human activity datasets," *IEEE Robotics and Automation Letters*, 2020.

[33] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," in *NeurIPS*, 2020.

[34] M. A. Lee, C. Florensa, J. Tremblay, N. Ratliff, A. Garg, F. Ramos, and D. Fox, "Guided uncertainty-aware policy optimization: Combining learning and model-based strategies for sample-efficient policy learning," in *ICRA*, 2020.

[35] Y. Lee, J. Yang, and J. J. Lim, "Learning to coordinate manipulation skills via skill behavior diversification," in *ICLR*, 2020.

[36] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, Feb. 1966.

[37] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *ICLR*, 2017.

[38] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," in *RSS*, 2017.

[39] P. Mandikal and K. Grauman, "Learning dexterous grasping with object-centric visual affordances," in *ICRA*, 2021.

[40] A. Mandlekar, F. Ramos, B. Boots, S. Savarese, L. Fei-Fei, A. Garg, and D. Fox, "Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data," in *ICRA*, 2020.

[41] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *AAAI*, 2016.

[42] K. Mo, L. Guibas, M. Mukadam, A. Gupta, and S. Tulsiani, "Where2act: From pixels to actions for articulated 3d objects," in *ICCV*, 2021.

[43] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *NeurIPS*, 2018.

[44] T. Nagarajan, C. Feichtenhofer, and K. Grauman, "Grounded human-object interaction hotspots from video," in *ICCV*, 2019.

[45] T. Nagarajan and K. Grauman, "Learning affordance landscapes for interaction exploration in 3d environments," in *NeurIPS*, 2020.

[46] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *ICRA*, 2018.

[47] S. Nasiriany, V. H. Pong, S. Lin, and S. Levine, "Planning with goal-conditioned policies," in *NeurIPS*, 2019.

[48] G. Neumann, C. Daniel, A. Paraschos, A. Kupcsik, and J. Peters, "Learning modular policies for robotics," *Frontiers in Computational Neuroscience*, 2014.

[49] M. Neunert, A. Abdolmaleki, M. Wulfmeier, T. Lampe, J. T. Springenberg, R. Hafner, F. Romano, J. Buchli, N. Heess, and M. Riedmiller, "Continuous-discrete reinforcement learning for hybrid control in robotics," in *CoRL*, 2019.

[50] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, *Solving rubik's cube with a robot hand*, 2019. arXiv: 1910.07113 [cs.LG].

[51] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, *Learning dexterous in-hand manipulation*, 2019. arXiv: 1808.00177 [cs.LG].

[52] T. Osa, J. Pajarinen, G. Neumann, J. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends in Robotics*, 2018.

[53] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *ICML*, 2017.

[54] D. Pathak, D. Gandhi, and A. Gupta, "Self-supervised exploration via disagreement," in *ICML*, 2019.

[55] K. Pertsch, Y. Lee, and J. J. Lim, "Accelerating reinforcement learning with learned skill priors," in *CoRL*, 2020.

[56] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *RSS*, 2018.

[57] J. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine, "Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings," in *ICML*, 2018.

[58] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[59] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman, "Dynamics-aware unsupervised discovery of skills," in *ICLR*, 2020.

[60] M. Sharma, J. Liang, J. Zhao, A. LaGrassa, and O. Kroemer, "Learning to compose hierarchical object-centric controllers for robotic manipulation," in *CoRL*, 2020.

[61] A. Simeonov, Y. Du, B. Kim, F. R. Hogan, J. Tenenbaum, P. Agrawal, and A. Rodriguez, "A long horizon planning framework for manipulating rigid pointcloud objects," in *CoRL*, 2020.

[62] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine, "Parrot: Data-driven behavioral priors for reinforcement learning," in *ICLR*, 2020.

[63] M. Smith, H. Hoof, and J. Pineau, "An inference-based policy gradient method for learning options," in *ICML*, 2018.

[64] R. Strudel, A. Pashevich, I. Kalevatykh, I. Laptev, J. Sivic, and C. Schmid, "Learning to combine primitive skills: A step towards versatile robotic manipulation," in *ICRA*, 2020.

[65] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," in *CoRL*, 2018.

[66] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Learning compositional models of robot skills for task and motion planning," *IJRR*, 2021.

[67] E. Wei, D. Wicke, and S. Luke, "Hierarchical approaches for reinforcement learning in parameterized action space," *arXiv*, vol. abs/1810.09656, 2018.

[68] F. Xia, C. Li, R. Martıín-Martıín, O. Litany, A. Toshev, and S. Savarese, "Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation," in *ICRA*, 2021.

[69] J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu, "Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space," *arXiv*, vol. abs/1810.06394, 2018.

[70] D. Xu, A. Mandlekar, R. Martín-Martín, Y. Zhu, S. Savarese, and L. Fei-Fei, "Deep affordance foresight: Planning through what can be done in the future," in *ICRA*, 2021.

[71] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese, "Neural task programming: Learning to generalize across hierarchical tasks," in *ICRA*, 2018.

[72] J. Yamada, Y. Lee, G. Salhotra, K. Pertsch, M. Pflueger, G. S. Sukhatme, J. J. Lim, and P. Englert, "Motion planner augmented reinforcement learning for obstructed environments," in *CoRL*, 2020.

[73] S. Zhang and S. Whiteson, "Dac: The double actor-critic architecture for learning options," in *NeurIPS*, 2019.

[74] Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín, "Robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv preprint arXiv:2009.12293*, 2020.

APPENDIX

A. Behavior Primitives

We elaborate on the manipulation primitives that we outlined in Section III-B: reaching, grasping, pushing, gripper release, and atomic. We classify all primitives that are not the atomic primitive as *non-atomic* primitives. Under the hood, all non-atomic primitives execute a variable sequence of atomic actions, either until the primitive is successfully executed or until a time limit is reached. All atomic actions specifically interface with the Operational Space Control (OSC) controller, which has 5 degrees of freedom: 3 degrees to control the position of the end effector, 1 degree to control the yaw angle, and (for all tasks but wiping) 1 degree to open and close the gripper.

We elaborate further on our non-atomic primitives. The gripper release primitive executes a fixed number of atomic actions to open the gripper. The reaching, grasping, and pushing primitives are hard-coded closed-loop controllers that all entail a reaching phase, either for reaching the starting location (for pushing) or for reaching the final location (for reaching and grasping). To implement this functionality for table-top environments (all except door), the robot first lifts its end effector to a pre-specified height, then hovers to the target XY position, and finally lowers its end effector to the target location. For other environments (door), the robot moves to toward the target location directory via the OSC controller. During this reaching phase, the reaching primitive keeps its gripper closed (except for the non-tabletop environments like door) and the grasping and pushing primitives keep their grippers open. The grasping and pushing primitives can be configured to achieve a specified yaw angle, which they satisfy during the reaching phase simultaneously while applying end effector displacements. Upon reaching, the grasping primitive emulates grasping by closing its gripper and the pushing the primitive emulates pushing by applying end effector displacements in a specified direction.

B. Algorithm

Our algorithm implementation is based on Soft Actor-Critic. Our algorithm alternates between collecting on-policy transitions in the environment and performing off-policy training on data sampled from the replay buffer. Training specifically entails optimizing the Q network, task policy, and parameter policy via gradient descent. As in the original SAC implementation, we use the reparameterization trick with respect to the parameter policy loss in order to reduce the variance of our gradient estimates. While we assume continuous primitive parameters we can also represent discrete parameters and apply reparameterization with the Gumbel-Softmax trick [25, 37]. We provide a full outline of our algorithm in Algorithm 1. Code is publicly available at `https://github.com/UT-Austin-RPL/maple`.

C. Affordance Score

We elaborate on the affordance score introduced in Section III-D:

$$s_{\text{aff}}(s, x; a) = \max_{p \in P} \; 1 - \tanh\Big(\max(\|x_{reach} - p\| - \tau, 0)\Big)$$

The keypoint p is dependent on the primitive a and the current state s. For example for the cleanup task, the keypoint for the pushing primitive is the location of a pushable object (the jello box), the keypoint for a grasping primitive is the location of a graspable object (the spam can), and the keypoint for the reaching primitive is the location of the bin. If there are multiple keypoints of interest we calculate the affordance score corresponding to each keypoint and consider the maximum score. If no applicable keypoint exists for a primitive (e.g. there are no pushable objects in door opening) we give an affordance score of 0. By default we set the threshold τ to 0.03 for grasping, 0.06 for reaching, and 0.12 for pushing. There are a few exceptions for tasks that need larger affordance regions for reaching large objects.

D. Flat Baseline

We considered two variants for our flat baseline. One variant, which has been explored by prior work [20, 68], outputs a distribution over the primitive type a and the parameters $\{x^1, x^2, \cdots, x^k\}$ for all primitives. As we discussed in Section III-C, under this approach the number of policy outputs scales linearly with the total number of primitive parameters, which can lead to optimization difficulties for large behavior libraries. Empirically we found this to be the case, as we were unable to make any progress on any of our tasks despite extensive hyperparamter tuning. Neunert et al. [49] proposed an alternative approach of replacing the distribution over all parameter outputs with the "one size fits all" distribution that we described in Section III-C. Parameter selection occurs by *independently* sampling a primitive type a and parameters x, and subsequently truncating the sampled parameters by the dimension of the sampled primitive type. This sampling strategy was also adopted by Lee et al. [35]. We note that this independent sampling process is in contrast to our two-stage hierarchical process. We adopted this variant as our flat baseline, and in Figure 4 we see that it often leads to sub-optimal performance. We hypothesize that this is due the fact that the parameter selection process is not informed by the primitive type selection process, which reduces the agent's utility especially when dealing with primitives that have heterogeneous parameter structures.

E. Open Loop Baseline

Our open loop baseline follows an open loop task schema, and is inspired from Chitnis et al. [8]. The open loop baseline and our method share identical implementations, except for the input to the task policy: our method takes in the current environment observation while the open loop baseline takes in only the current episode timestep. We highlight that while our implementation is inspired from Chitnis et al. [8], there are notable differences. Their update rule for

Algorithm 1 **Ma**nipulation **P**rimitive-augmented reinforcement **Le**arning (MAPLE)

Initialize Q network $Q_\theta(s, a, x)$, task policy $\pi_{tsk_\phi}(a|s)$, parameter policy $\pi_{p_\psi}(x|s, a)$, replay buffer $\mathcal{D}$
for iteration $1, \ldots, N$ **do**
  for episode $1, \ldots, M$ **do** {Exploration Phase}
    Initialize timer $t \leftarrow 0$
    Initialize episode s_0
    while episode not terminated **do**
      Sample primitive type a_t from task policy $\pi_{tsk_\phi}(a_t|s_t)$
      Sample primitive parameters x_t from parameter policy $\pi_{p_\psi}(x_t|s_t, a_t)$
      Truncate sampled parameters to dimension of sampled primitive $x_t \leftarrow x_t[: d_{a_t}]$
      Execute a_t and x_t in environment, obtain reward r_t and next state s_{t+1}
      Add affordance score to reward $r_t \leftarrow r_t + \lambda s_{\text{aff}}(s_t, x_t; a_t)$
      Add transition to replay buffer $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, a_t, x_t, r_t, s_{t+1}\}$
      Update timer $t \leftarrow t + 1$
    end while
  end for
  for training step $1, \ldots, K$ **do** {Training Phase}
    Update Q network: $\theta \leftarrow \theta - \lambda_{lr} \nabla_\theta J_Q(\theta)$
    Update task policy: $\phi \leftarrow \phi - \lambda_{lr} \nabla_\phi J_{\pi_{tsk}}(\phi)$
    Update parameter policy: $\psi \leftarrow \psi - \lambda_{lr} \nabla_\psi J_{\pi_p}(\psi)$
  end for
end for

the "task policy" (or equivalent thereof) does not use gradient descent, relies on on-policy sampling, and is designed for the sparse reward setting only. We found these assumptions to be restrictive for our algorithmic and task setup, and we instead use gradient-based, off-policy reinforcement learning methods which can work with sparse or dense rewards. Despite these differences, we believe that our open loop baseline captures the essence of the ideas proposed in Chitnis et al. [8] – namely that open loop task shemas can enable more efficient and effective learning. As we show in Figure 4, however, we did not find this to be the case for the relatively more complex tasks in our suite of manipulation domains.

F. DAC Baseline

We considered a number of potential methods as our representative option-learning baseline. While prominent prior work [4, 31, 73] has focused on learning options, we verified that Double Actor-Critic (DAC) achieves superior performance on the OpenAI HalfCheetah-v2 task and our lift task, so we chose DAC as our representative options baseline. We also considered Deep Skill Chaining [5], another recent work that learns options; however it was not applicable to our manipulation domains given that it is designed primarily for goal-based navigation agents. We used the implementation publicly released by the DAC authors [1], and we adopted the hyperparameters that they suggested in their paper.

G. Environments

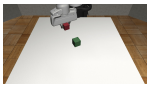
We conduct experiments on eight manipulation tasks of varying complexities, spanning diverse prehensile and non-prehensile behaviors. The first six come from the standard robosuite benchmark [74]. We designed the last two (cleanup, peg insertion) to evaluate our method in multi-stage, contact-rich tasks. We elaborate on each as follows:

Lift: the robot must pick up a cube and lift it above the table.

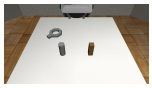
Door Opening: the robot must turn the door handle and open the door.

Pick and Place: the robot must pick up a soda can and place it into a specific target compartment.

Wipe: the robot must wipe a table containing spilled debris. A penalty is given if the robot presses too hard into the table.



Stack: the robot must stack a cube on top of another cube.



Nut Assembly: the robot must fit a nut tool onto the round peg.

Cleanup: the robot must store a spam can into a storage bin and store a jello box at the upper right corner.

Peg Insertion: the robot must pick up the peg and insert it into the opening of a wooden block.

H. Training

We provide a full list of our algorithm hyperparameters in Table I. We note a few additional details. For a consistent comparison across baselines, our episode lengths are fixed to 150 *atomic* timesteps, meaning that we execute a variable number of primitives until we have exceeded the maximum number of atomic actions for the episode. Also, for the first 600k environment steps we set the target entropy for the task policy and parameter policy to a high value to encourage higher exploration during the initial stages of training.

I. Evaluation

We elaborate on the evaluation protocol for our experiments in Figure 4. We evaluate each experimental variant

[1] https://github.com/ShangtongZhang/DeepRL/tree/DAC

TABLE I: Hyperparameters for our algorithm

Hyperparameter	Value
Hidden sizes (all networks)	$256, 256$
Q network and policy activation	ReLU
Q network output activation	None
Policy network output activation	tanh
Optimizer	Adam
Batch Size	1024
Learning rate (all networks)	3e$-$5
Target network update rate τ	1e$-$3
# Training steps per epoch	1000
# (Low-level) exploration actions per epoch	3000
Replay buffer size	1e6
Episode length (# low-level actions)	150 (except wipe, 300)
Discount factor	0.99
Reward scale	5.0
Affordance score scale λ	3.0
Automatic entropy tuning	True
Target Task Policy Entropy	$0.50 \times log(k)$, k is number of primitives
Target Parameter Policy Entropy	$-\max_a d_a$

(combination of task and method) over 5 seeds and we (1) plot the agent's rewards throughout training and (2) report the task success rate at the end of training. Specifically for the reward plots, we evaluate the agent's average episodic task rewards (excluding the affordance reward) at regularly spaced training checkpoints every $30k$ environment exploration steps. The episodic rewards are averaged over 20 episodes and are normalized between 0 and 100, where 100 corresponds to the agent receiving the maximum possible reward at every single timestep of the episode. We post-process the plots, showing the moving average of results over the last $150k$ environment steps. For reporting the final task success rate, we load the final training checkpoint and report the average task success rate over 20 episodes. Success rates for our tasks are defined as follows:

- Lift: whether the block is above a height threshold
- Door: whether the door angle is past a threshold
- Pick and Place: whether the can is in the correct target bin and the robot is not holding the can
- Wipe: whether all of the debris is wiped off the table
- Stack: whether the smaller cube is on top of the larger cube and the robot is not holding either cube
- Nut Assembly: whether the nut is fitted completely onto the round peg and the robot is not holding the nut
- Cleanup: whether the spam can is in the bin and the jello box is within a threshold distance away from the table corner
- Peg Insertion: whether the peg is inserted into wooden block past a threshold distance

Final task success rates for all baselines across all tasks are outlined in Table II.

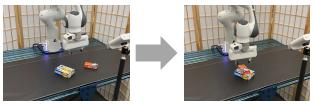
We also elaborate on the evaluation protocol for the compositionality scores that we report in Figure 5. Our compositionality scores are averaged over 5 seeds for each task. For each seed we sample 50 task sketches from the last training checkpoint and we discard sketches that did not correspond to the agent solving the task. Of these remaining task sketches we calculate the compositionality score according to Equation (1).

J. Task Sketch Transfer Experiments

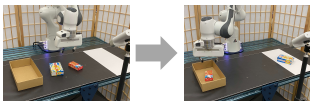
For our task sketch experiments we first extract a set of task sketches $\{K_1, \cdots, K_n\}$ from the source task. We subsequently select the sketch K that has the lowest Levenshtein distance with all other task sketches. In the case of our pick and place task this was $\{$Grasp, Reach, Release$\}$. Once we have extracted the sketch K from the source task, we train on the target task with a fixed task sketch of K. For each episode we iterate through the sequence of primitives in K, repeating each primitive up to 5 times until the agent receives high affordance reward, before moving onto the next primitive in the sketch. Upon executing all of the primitives in the task sketch the agent executes 10 atomic primitives to satisfy any behaviors that it was not able to fulfill with the sketch alone, and then the episode terminates.

K. Real-World Experiments

We performed evaluations on two real-world manipulation tasks:



Stack: the robot must pick up the butter box and stack it on top of the popcorn box.



Cleanup: the robot must (1) pick up the butter box and place it into the bin and (2) push the popcorn to the right side of the table (the white area).

Both tasks resemble the simulated stack and cleanup tasks outlined in Appendix G, but have differences in the size of the objects, table size, and workspace layout. To account for these differences we designed variations of our existing simulated stack and cleanup tasks to match the characteristics of our real-world tasks. We trained policies in simulation (until convergence) and transferred them to the real-world for evaluation.

TABLE II: Final Task Success Rates (%)

	Lift	**Door**	**Pick and Place**	**Wipe**	**Stack**	**Nut Assembly**	**Cleanup**	**Peg Insertion**
Atomic [19]	**98.0 ± 2.4**	0.0 ± 0.0	0.0 ± 0.0	18.0 ± 18.3	38.0 ± 28.7	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Flat [35, 49]	61.0 ± 47.8	**100.0 ± 0.0**	1.0 ± 2.0	22.0 ± 9.8	**98.0 ± 2.4**	0.0 ± 0.0	0.0 ± 0.0	8.0 ± 13.6
Open Loop [8]	43.0 ± 43.5	**100.0 ± 0.0**	81.0 ± 38.0	16.0 ± 4.9	85.0 ± 3.2	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
DAC [73]	75.0 ± 12.7	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	16.0 ± 32.0
MAPLE (Non-At.)	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**42.0 ± 9.8**	**99.0 ± 2.0**	93.0 ± 14.0	**100.0 ± 0.0**	0.0 ± 0.0
MAPLE (ours)	**100.0 ± 0.0**	**100.0 ± 0.0**	**95.0 ± 7.7**	**42.0 ± 11.7**	**98.0 ± 2.4**	**99.0 ± 2.0**	91.0 ± 5.8	**100.0 ± 0.0**

For perception we use the robot proprioception data from the robot's on-board sensors, in conjunction with pose estimates of the objects using the deep object pose estimation system [65] paired with a single Microsoft Kinect camera. Under this setup the objects are sometimes out of the camera view or are occluded by the robot arm; in these cases the pose estimator does not return estimates of the object. We mostly alleviate such conditions through a three step procedure: (1) the robot lifts its end-effector to a pre-determined location in the air where occlusions are minimized, (2) the pose estimator re-computes the poses of the objects, (3) the robot moves back to its initial location. At the end of step (3), the pose of objects are assumed to be the pose estimates from step (2), with the exception of objects that were moved by the robot during step (1). Such objects comprise objects that the robot was already holding before step (1) and that the robot subsequently lifted into the air during step (1). For such objects, we compute the pose of the object as the final robot pose in addition to the relative pose difference of the robot end effector and object during step (2). In addition to handling occlusions, we noticed that the pose estimation system routinely made small errors when estimating the position and orientation of the objects. To minimize the influence of such errors, we hard-coded the pitch and yaw angles of the objects (as they were always flat either on the table or in the air) and the height of the object whenever it was detected to lie on the table. These constraints were necessary to ensure reliable perception estimates, but we anticipate that with improved perception systems in the future such constraints can be relaxed.

For evaluation, we performed 30 trials for each task, where the robot was allowed a maximum of 20 primitive calls per episode. We recorded an average success rate of 93% for stack (in contrast to 97% in simulation), and 83% for cleanup (in contrast to 93% in simulation). Most failures were either due to the robot repeatedly applying poor grasping actions or the robot hitting its joint limits, subsequently triggering a safety call to halt the robot.